The Ties That Guide Us (incursion)

Now that you have hired an assistant using the profits from your robot sales, you are finally ready to go for the safe that contains the CEOI medals!

The safe is located in a university building that consists of N rooms with $N - 1$ doors connecting them. Every room can be reached from any other, and each room has at most 3 doors. Both you and your assistant have a floor plan of the building—but unfortunately in two different editions: While both plans describe the same layout, *the numbering of the rooms and doors might differ*.

During the second competition day, the committee will be busy handling clarification requests. This is the perfect opportunity to get to the safe with the medals.

In the beginning, your assistant will search the building. Once he has found the room with the safe, he will guide you to it.* As you are not allowed to bring your phone to the competition venue, your assistant leaves behind secret signs in the university rooms. For this, he uses the virtually infinite supply of ties left over from last year's BOI. Because these ties are *indistinguishable*, the only information you will get is the number of ties left behind by your assistant in any given room. As too many ties in one place might look suspicious, the *maximum number of ties in any individual room should be as small as possible* (see the grading section below).

Later, you will sneak out during a toilet break and use the ties left behind to find the room with the safe. Beware that the safe will of course be hidden, so you will not be able to recognize it just by entering the room and instead will have to rely on the ties you've found. Moreover, as your absence must not be noticed, you have to find the safe quickly: *you should pass through at most $d + 30$ doors, where d is the number of doors on the direct path from your starting position to the room with the safe.* If you pass through a door multiple times, each pass counts.

Write a program that

- tells your assistant how many ties to leave behind in each room, *and*
- guides you to the room with the safe afterwards.

Communication

This is a communication task in which your program is run *twice* for each testcase. You must implement the following two functions:

- **vector**⟨**int**⟩ *mark*(**vector**⟨**pair**⟨**int**, **int**⟩⟩ F, **int** *safe*) where
  - F describes the floor plan by specifying the doors: F contains $N - 1$ pairs (u, v) with $1 \le u, v \le N$ and $u \ne v$, which means that there is a door between rooms u and v;
  - *safe* is the room with the safe.

  Your function should return a vector T that consists of N integers $T[0], \ldots, T[N-1]$ with $0 \le T[i] \le 10^9$: $T[i]$ is the number of ties your assistant leaves behind in room $i + 1$. *The numbers $T[i]$ should be as small as possible.*
- **void** *locate*(**vector**⟨**pair**⟨**int**, **int**⟩⟩ F, **int** *curr*, **int** t) where F describes the same floor plan (but possibly with a different numbering of the rooms and a different ordering of the doors!), *curr* is the room you are currently in, and t is the number of ties you found there.

  Inside this function you can call the grader function **int** *visit*(**int** v) in order to move from your current room to room v (according to the room numbering in your version of the floor plan,

* After all, you want to open the safe yourself—it would be no fun to have someone else open it and hand you the medal after the competition!

$1 \le v \le N$); this function returns the number of ties your assistant left behind in room v. There must be a door connecting room v to your current room, and you can only call this function a limited number of times per testcase (see above).

When your function *locate* returns, you have to be in the room with the safe.

For each testcase, in the first run the function *mark* is called, while in the second run *locate* is called.

If you call *visit* too often, with invalid parameters, or during the first run of your program, your submission will be immediately terminated and judged as **Not correct** for the corresponding testcase. You must not write to standard output or read from standard input; otherwise, you may receive the verdict **Security violation!**. You are however free to write to the standard error stream (*stderr*).

You must include the file `incursion.h` in your source code. To test your program locally, you can link it with `sample_grader.cpp` which can be found in the attachment for this task in CMS. See below for a description of the sample grader, and see `sample_grader.cpp` for instructions on how to run it with your program. Beware that for simplicity this sample grader does *not* run your program twice, but instead calls both *mark* and *locate* (exactly once) as part of a single run. The attachment also contains a sample implementation as `incursion_sample.cpp`.

Constraints and grading

We always have $2 \le N \le 45\,000$.

Subtask 1 (30 points). There is no room with 3 doors.

Subtask 2 (30 points). There is precisely one room with 2 doors.

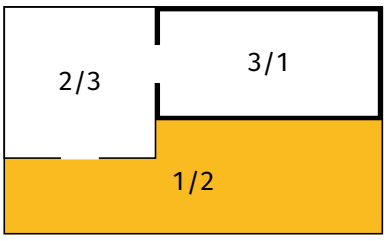
Subtask 3 (40 points). No further constraints.

Partial scoring. In all subtasks, your actual score depends on the maximum number $T_{\max}$ of ties left behind by your assistant in a room (i.e. the largest number $T[i]$ in the return value of *mark*) according to the following table:

$T_{\max}$	$[0, 1]$	2	$[3, 10^9]$
score	100%	40%	30%

Example interaction

Consider a testcase with $N = 3$ and the following floor plan:



The first number in each room refers to your assistant's numbering and the second one to your own numbering. The safe is located in the large room at the bottom (shaded in yellow) while your starting location is in the upper right room (drawn with a bold outline).

In the first run of your program, the grader calls your function *mark* as *mark*({{1, 2}, {2, 3}}, 1). Assume this returns {2, 2, 0}, i.e. your assistant leaves behind two ties in rooms 1 and 2 each, but no tie in room 3.

In the second run of your program, the grader calls your function *locate* as *locate*({{1, 3}, {2, 3}}, 1, 0). Then, an interaction between your program and the grader could look as follows:

Your program	Return value	Explanation
visit(3)		you visit room 3 in your numbering
	2	this is room 2 in your assistant's numbering, in which he left behind two ties
visit(1)		(re)visit your room 1...
	0	... corresponding to your assistant's room 3, which contains no ties
visit(3)		(re)visit your room 3...
	2	... which still has two ties
visit(2)		visit your room 2...
	2	... corresponding to your assistant's room 1, with two ties
return		you are sure that room 2 contains the safe as this corresponds to room 1 in your assistant's numbering, your program is judged correct

Here you passed through four doors, while the number d of doors on the direct path from your starting location to the room with the safe is two.

This interaction is reproduced by `incursion_sample.cpp` on the public testcase.

Grader

The sample grader first expects on standard input the integers N and *safe*. Afterwards, it expects the floor plan F as a sequence of $N - 1$ pairs of integers $1 \le u, v \le N$ ($u \ne v$).

It will then call *mark*(F, *safe*) and write its return value T to standard output. After this, it expects your starting location *curr* and will call *locate*(F, *curr*, T[*curr* – 1]); *note that the sample grader will not change the numbering of rooms and doors.* It will then write to standard output a protocol of all calls to *visit* by your program.

Upon termination, the sample grader writes one of the following messages to standard output:

Invalid input. The input to the grader via standard input was not of the above form.

Invalid call to visit. You called *visit* from *mark* or with an invalid parameter.

Invalid return value of mark. The call to *mark* did not return a vector T of length N or at least one of the values $T[i]$ was either negative or larger than 10^9.

Correct: at most T_{max} tie(s) per room. The function *locate* returned while in room *safe*, and the largest value in T is T_{max}.

Not correct: current position is *curr*. The function *locate* returned while in room *curr* ≠ *safe*.

In contrast, the grader actually used to judge your program will only output **Not correct** (for any of the above errors), **Security violation!**, **Partially correct**, or **Correct**. Moreover, the grader is *adaptive*, i.e. its behavior may depend on the actions of your program in the current as well as in earlier runs. Both the sample grader and the grader used to judge your program will terminate your program automatically whenever one of the above errors occurs.

Limits

Time: 0.5 s
Memory: 512 MiB

For every testcase, the above time and memory limit will be enforced for each of the two runs of your program *separately*.